SOFTWARE DEVELOPMENT II

OVERALL PROJECT REPORT:

Reengineering the legacy code of the latex editor project

TEAM

Antoniou Christodoulos AM: 2641
Tsiouri Angeliki AM: 3354

TABLE OF CONTENTS

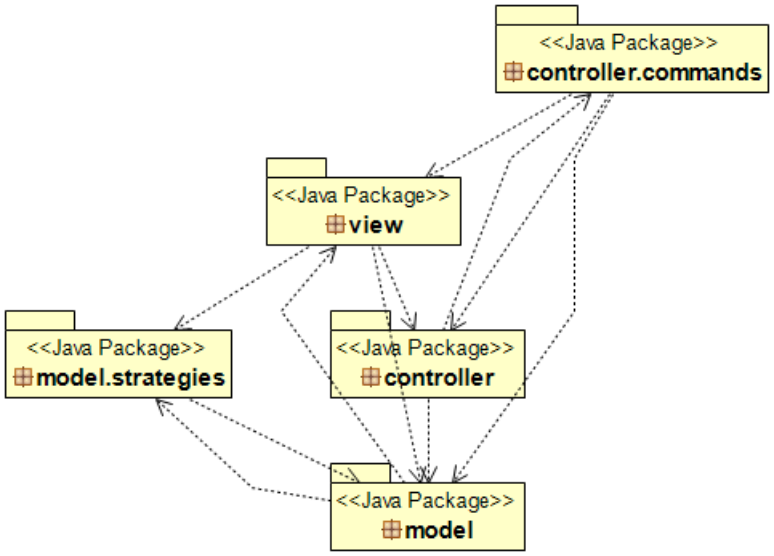
SUMMARY

The goal of the project was to reengineer and extend an existing Java application. The application is a simple Latex editor for inexperienced Latex users. The first phase of the project revolved around understanding the application's legacy architecture. We studied the documentation, user stories and source code and set up a mock installation of the application. Moreover, test cases were prepared for each of the user stories and the design was captured in UML package and class diagrams. In the second phase refactoring and extension was performed, aiming to improve code quality and extensibility as well as usability for both developers and users. Refactoring techniques such as **Extract Method**, **Move Method**, **Substitute Algorithm**, **Singleton Pattern**, **Remove Middle Man** and **Externalize Strings** were used to address issues such as **Duplicate Code**, **Long Method**, **Dead Code**, **Message Chains** and **Misplaced Responsibilities**. The application was also extended to support two more user stories regarding HTML document import and export. The refactored design was captured in UML package and class diagrams as well as CRC cards.

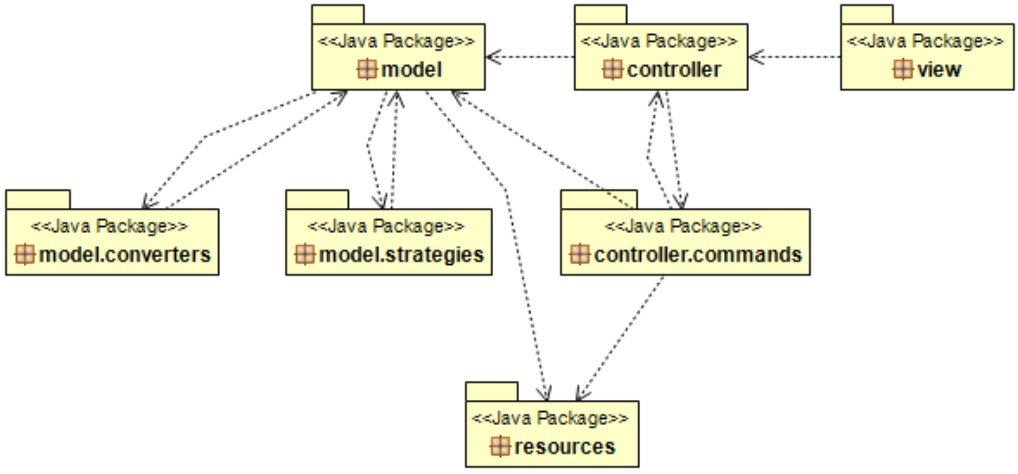
THE LATEX EDITOR PROJECT

Latex Editor is a simple editor aiming to assist inexperienced Latex users with Latex document preparation. Latex is a high-quality document preparation markup language. It provides a large variety of styles and commands that enable advanced document formatting. Formatting documents with Latex is similar to a programming process as it involves the proper usage of Latex commands which are embedded in the document contents. The goal of the Latex Editor is to facilitate the usage of Latex commands for the preparation of Latex documents. One of the prominent features that distinguishes the Latex Editor from other similar applications is its multi-strategy version tracking functionality that enables undo actions.

ARCHITECTURE



Application architecture prior to reengineering



Application architecture after reengineering

DETAILED DESIGN – REFACTORING TASKS

In this section, the implemented refactors are explained in detail for each class followed by the respective UML class diagrams for the classes of each package.

LatexEditorController class

A problem here was **Duplicate Code** in the constructor of the class. The same commands were repeated to populate the HashMap with the commands of the application. The issue was refactored using the **Substitute Algorithm**. The string command names are now stored in a list class field. The list is iterated in the createCommands() method to create the required command objects.

Command classes

An issue of **Duplicate Code** was also existing among the command classes. However, in this case the issue was in the form of common objects as class fields among the command classes. The issue was solved by applying the **Singleton Pattern** to provide single points of access to the classes of the key objects used by the commands. More specifically, the Singleton Pattern has been applied to the VersionTrackingManager, DocumentManager and LatexEditorController classes. Moreover, unnecessary methods such as the getter and setter in the LoadCommand class have been removed.

DocumentManager class

In this class there were two issues. The first one was **Duplicate Code** in the constructor of the class, similar to the issue of the LatexEditorController class. The issue was refactored using the **Substitute Algorithm** in the same manner as the one of the LatexEditorController class. The string template names are now stored in a list class field. The list is iterated in the constructor to create the required document objects. The second issue was a **Long Method**. In particular, getContents() was a huge method with a clear **Duplicate Code** problem. The issue was partly caused due to multi-line strings with document template contents which were hard-coded into the method. The strings were **externalized to a properties file** in a newly created **resources package**. A **Strings class** was created in the resources package which is responsible for fetching the corresponding strings. In that way it was possible to fetch any string by providing its template type and therefore the duplicate code issue was solved. It is now also much easier to potentially add more document template types in the future. Finally, the getContents() method was removed since it proved to be an **unnecessary method**. Instead, it is sufficient to fetch each string of template contents once in the constructor of the class and create one Document object for each template type.

VersionsManager class

This class has been **renamed to VersionTrackingManager** to match its role more appropriately. Naturally, any other references to the class in the source code have been renamed too. The simplest problem in this class was **Dead Code**. Unused methods have been removed. Other problems were **Misplaced Responsibilities** regarding saving and loading document to/from files as well as **Message Chains**. VersionsManager was a middle man with several methods (getType(), saveContents(), saveToFile(), loadFromFile(), etc.) that simply delegate invocations to corresponding methods of the LatexEditorView class. The delegating methods were removed using the **Remove Middle Man** refactoring. The delegate class LatexEditorView object was accessed from the server class objects and calls to delegating methods were replaced with direct calls to methods in the delegate class. Note that LatexEditorView was also removed so the above does not describe the final reengineering result.

LatexEditorView class

This class had critical issues. It was part of the view package which contains classes that have to do with the GUI and data visualization. LatexEditorView does not have anything to do with these. Instead, it contained application logic with methods that realize basic commands such as saving or loading a document and creating a new version of the current document. The issues were solved using the **Move Method** and **Move Field** refactorings to redistribute the responsibilities to the right classes. Specifically, the following refactorings were done:

- Rename VersionsManager field to VersionTrackingManager and move to LatexEditorController.
- Rename type field to templateType and move to LatexEditorController.
- Move currentDocument field to DocumentManager.
- Move filename field to DocumentManager.
- Rename strategy field to strategyType and move to VersionTrackingManager.
- Move saveContents() method to EditCommand.
- Move loadFromFile() method to LoadCommand.

In addition, saveToFile() method was removed since it was a **Message Chain** that delegated invocation to the save() method of the Document class. The issue was dealt with using the Remove Middle Man refactoring, similarly to the case of the VersionTrackingManager class. Moreover, methods readFileContents() and findTemplateType() were **extracted** from loadFromFile() method and placed in LatexEditorController. The remaining code of loadFromFile() was placed back to LoadCommand. Any remaining methods of LatexEditorView class were moved to LatexEditorController and finally LatexEditorView was removed.

MainWindow class

A problem here was a **Large Method** with **Duplicate Code** which was also a **Misplaced Responsibility**. More specifically, the editContents() method is part of the application logic and was moved to the AddLatexCommand class. The issues were caused by multi-line strings with the corresponding Latex commands which were hard-coded into the method. The strings were **externalized to a properties file** in the **resources package**. The Strings class is responsible for fetching the corresponding strings. In that way it was possible to fetch any string by providing its command type and therefore the duplicate code issue was solved, similarly to the case of the DocumentManager class. It is now also much easier to potentially support more Latex command types in the future. Another **Large Method** here was the initialize() method which is responsible for initializing the various components of the GUI. The method was relieved using the **Exctract Method** refactor such that each GUI component (JMenuItem, JMenu, JMenuBar, etc.) is initialized in its own method. Moreover, the values that determine positions and dimensions of frames and components were converted to static final class fields. In that way it is much easier to potentially adjust or extend the application's GUI in the future. Note that many of the components of the GUI were converted to class fields with respectable getters to support the development of test cases for the application.

OpeningWindow & ChooseTemplate classes

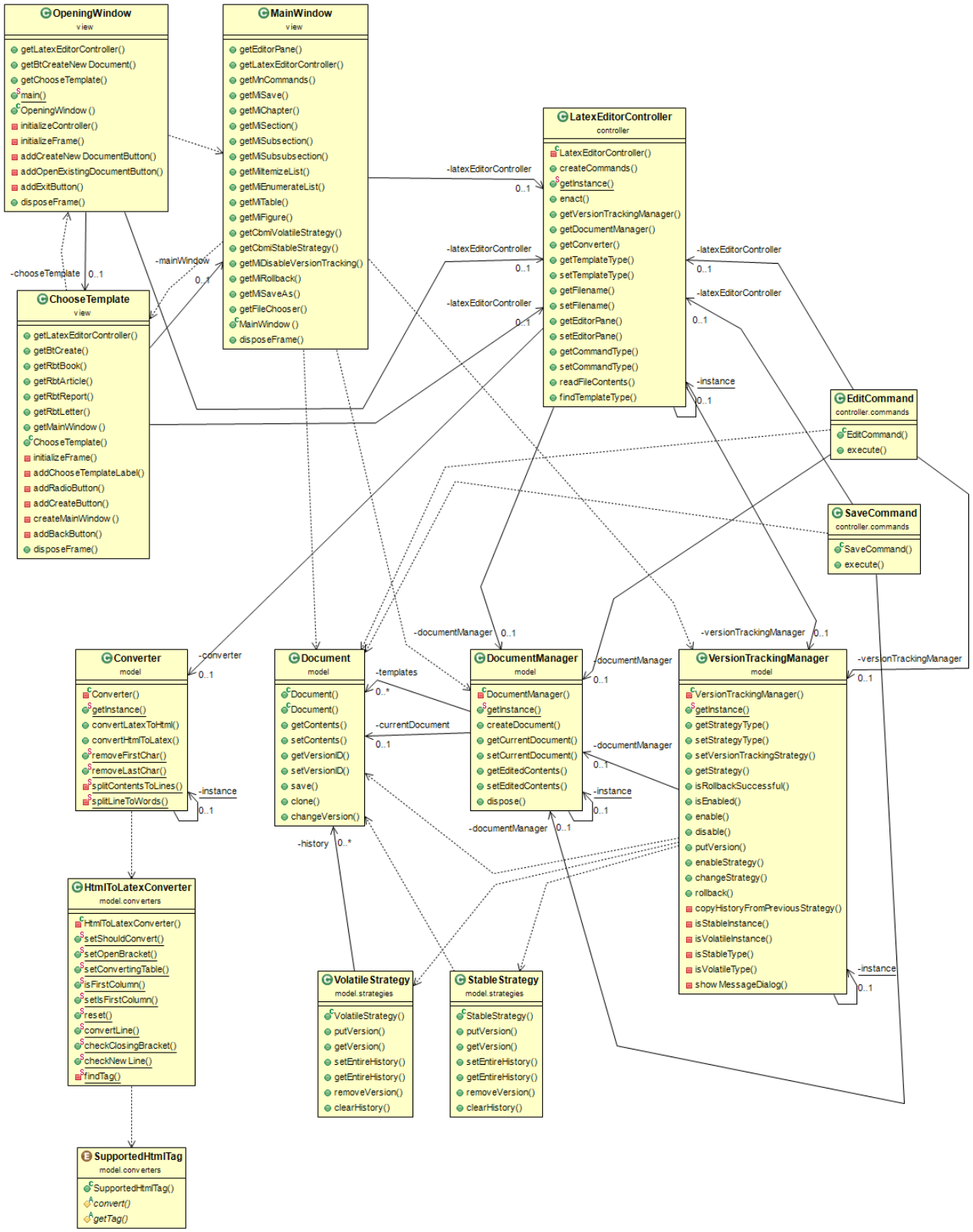
The above classes were refactored in a similar manner to the MainWindow class. Both classes had an issue with their initialize() methods being **Large Methods**. . The methods were relieved using the **Exctract Method** refactor such that each GUI component (JMenuItem, JMenu, JMenuBar, etc.) is initialized in its own method. Moreover, the values that determine positions and dimensions of frames and components were converted to static final class fields. In that way it is much easier to potentially adjust or extend the application's GUI in the future. Note that many of the components of the GUI were converted to class fields with respectable getters to support the development of test cases for the application.

Other refactors and fixes

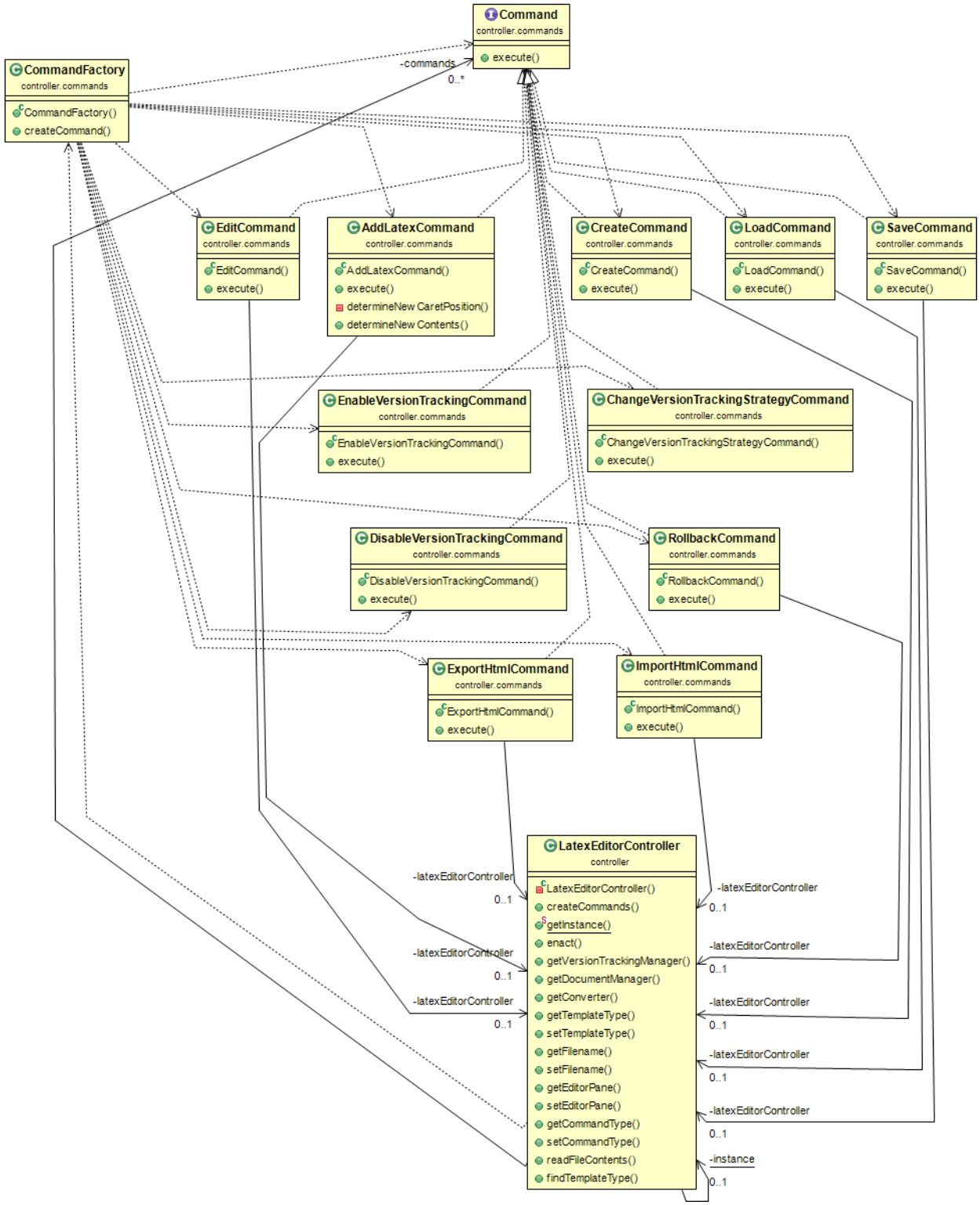
In addition to the above refactors, we attempted to further improve the code quality throughout the source code as well as the user experience. Some of the general guidelines that were followed include: Methods should rarely, if ever, surpass 20 lines of code. Field, method and class names should demonstrate their responsibilities as clearly as possible. Lines of code should rarely, if ever, surpass 100 characters in length. Code maintenance and extensibility should be as easy as possible. The purpose of different GUI components should be as clear as possible for the end user.

The following changes were implemented:

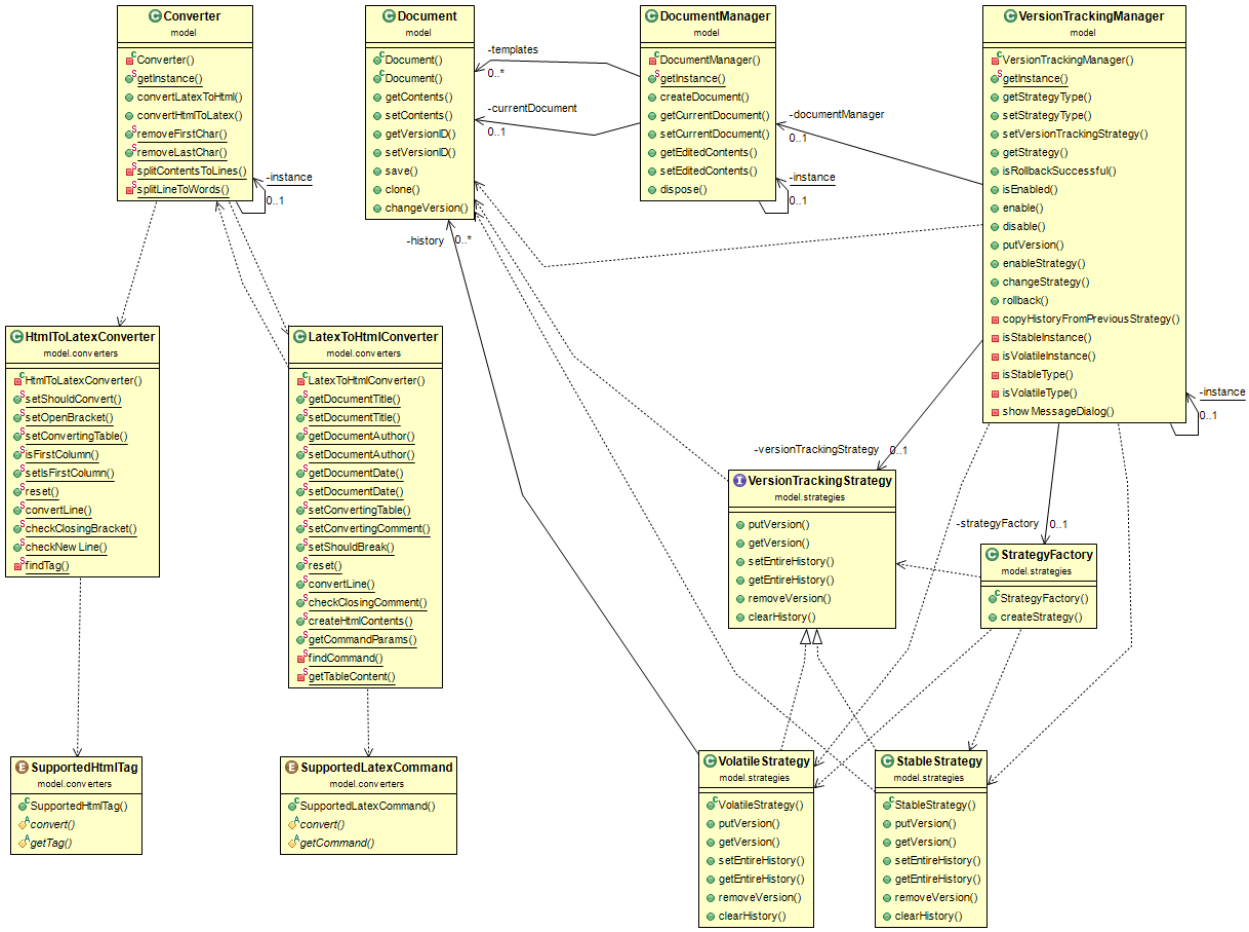
- Refactor duplicate code for latex command menu items initialization in MainWindow.
- Refactor duplicate code for check boxes initialization in MainWindow.
- Use ButtonGroup to manage RadioButton selection in ChooseTemplate.
- Refactor duplicate code and unused methods in ChooseTemplate.
- Add separators for MenuBar MenuItems in MainWindow.
- Adjust MenuBar MenuItems' names in MainWindow to clearly demonstrate their purposes.
- Refactor and use StrategyFactory (it was unused).
- Refactor duplicate code in VersionTrackingManager.
- Remove unnecessary/misleading comments throughout the source code.
- Fix open existing file functionality in OpeningWindow.
- Fix exit on close for OpeningWindow and ChooseTemplate GUIs.
- Adjust disabled command MenuItems in MainWindow such that when a template type does not support a command, only the corresponding command's MenuItem is disabled.
- Organized version tracking files (from the stable version tracking strategy) in a respective folder.



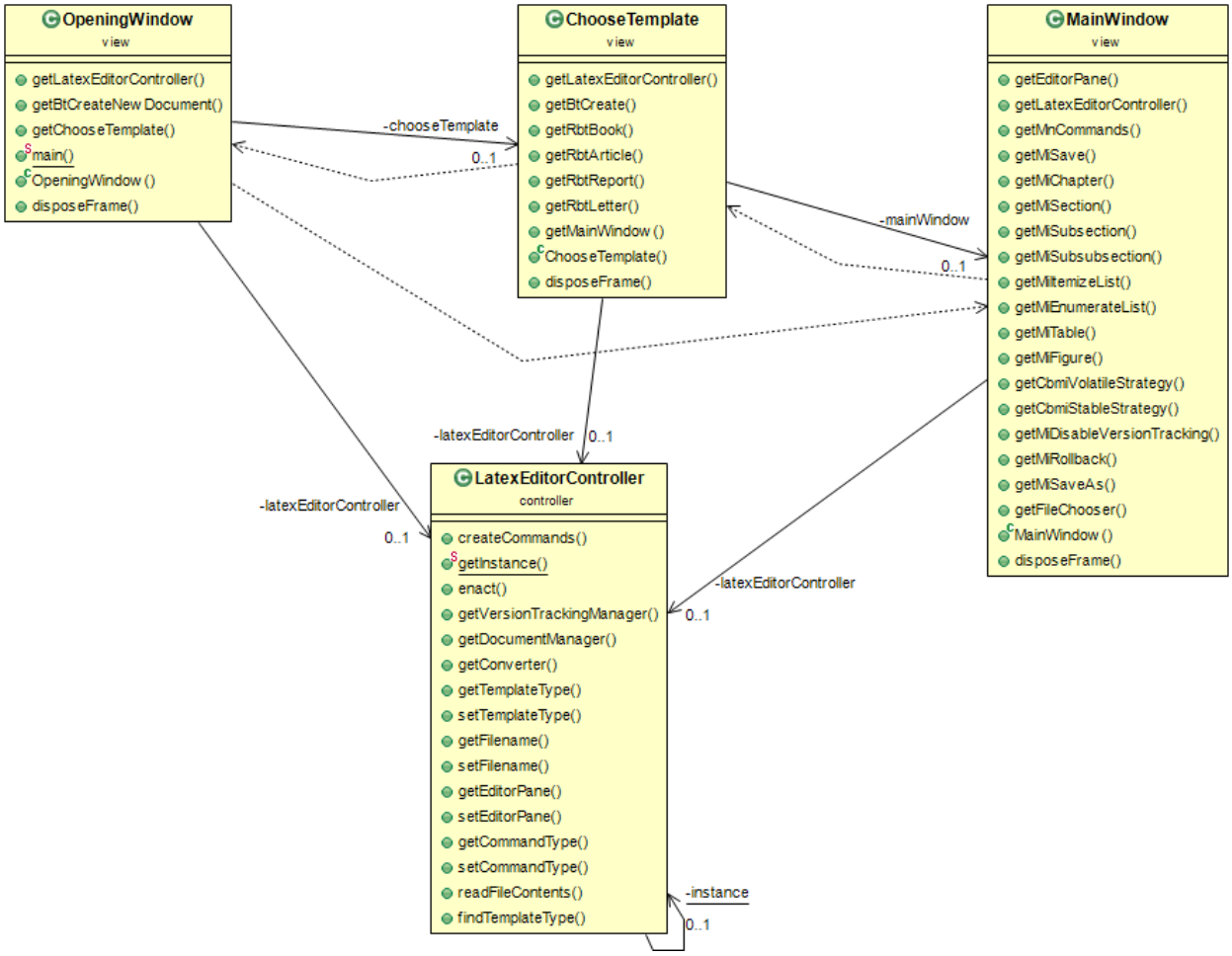
UML class diagram which outlines the general functionality of the application's architecture



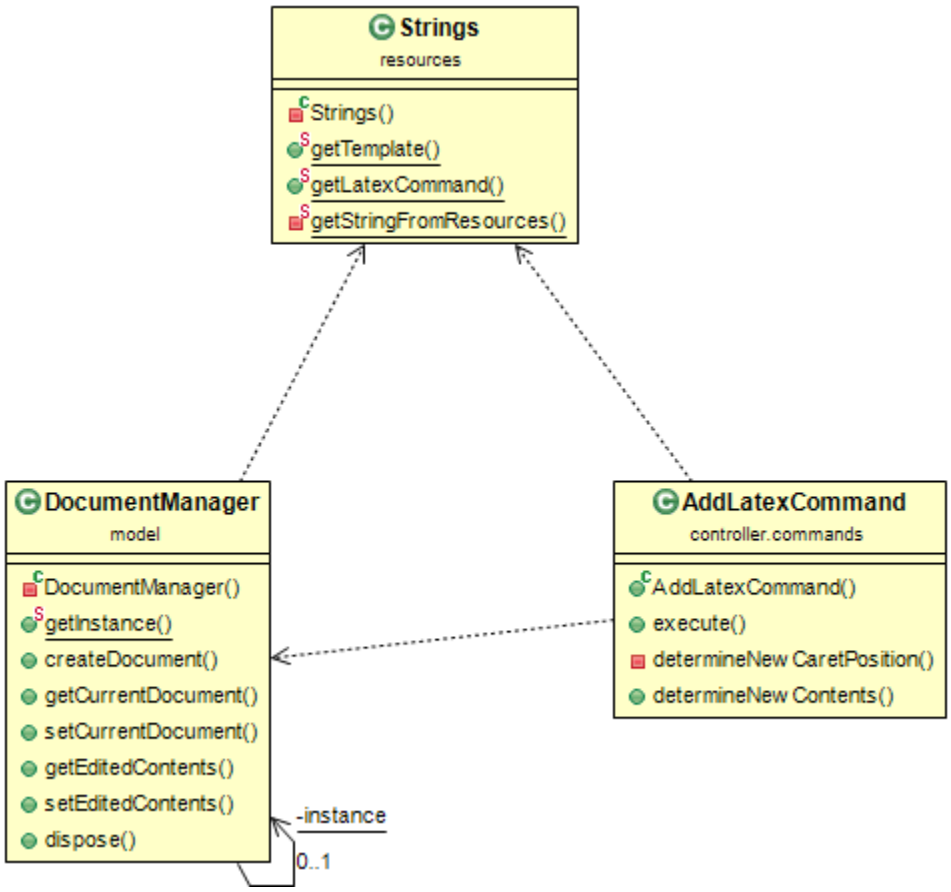
UML class diagram for the classes of the controller package



UML class diagram for the classes of the model package



UML class diagram for the classes of the view package (including LatexEditorController)



UML class diagram regarding strings fetching

DETAILED DESIGN – EXTENSION TASKS

The application was extended to support two more user stories, regarding document import from HTML files (conversion from HTML to Latex and loading) as well as document export to HTML (conversion from Latex to Html and saving). For this purpose, respective Import -> HTML and Export -> HTML MenuItems were added in the File Menu of the MenuBar of the MainWindow GUI.

The following classes and enums were developed:

Converter class

This is the main class responsible for any kind of document conversion. For the purposes of the project, it supports conversion from Latex to HTML and conversion from HTML to Latex for a specific set of commands/tags. The conversions are implemented by the respective methods, convertLatexToHtml() and convertHtmlToLatex() which split the document contents into words that are converted using methods of the HtmlToLatexConverter or LatexToHtmlConverter classes. The Converter class contains a few more extracted helper methods regarding string manipulation. The class is developed to support potential addition of other conversions by adding an extra method for each conversion. Also, the Singleton Pattern is implemented for the class.

SupportedHtmlTag & SupportedLatexCommand enums

These enums contain the HTML tags and Latex commands which the application supports for conversion. Each tag/command is represented by a value in the respective enum. Each enum value overrides two methods. A getter method which corresponds to the tag/command to be converted as well as a convert method which handles the conversion of the specific tag/command. The use of enums proved to be the best approach since it allows to store all tags/commands in the same place. It also supports potential application extension to support more tags/commands for conversion, it allows for easy adjustments to the existing values and it prevents duplicate code (multiple if statements) in other classes that use the enums. The enum files are placed in a newly created sub-package called model.converters.

HtmlToLatexConverter class

This class is also placed in the model.converters sub-package. Its main purpose is to convert a line from HTML to Latex, oftentimes taking into consideration factors that might come from another line. For instance, during table conversion. The main method of the class is convertLine() which, as its name suggests, is called from the Converter class and converts the

strings of a line from HTML to Latex. The class has a helper method which iterates over the SupportedHtmlTag enum values to find whether a string corresponds to an HTML tag. Moreover, the class has helper methods and boolean fields, regarding conversion of brackets, new lines and tables.

LatexToHtmlConverter class

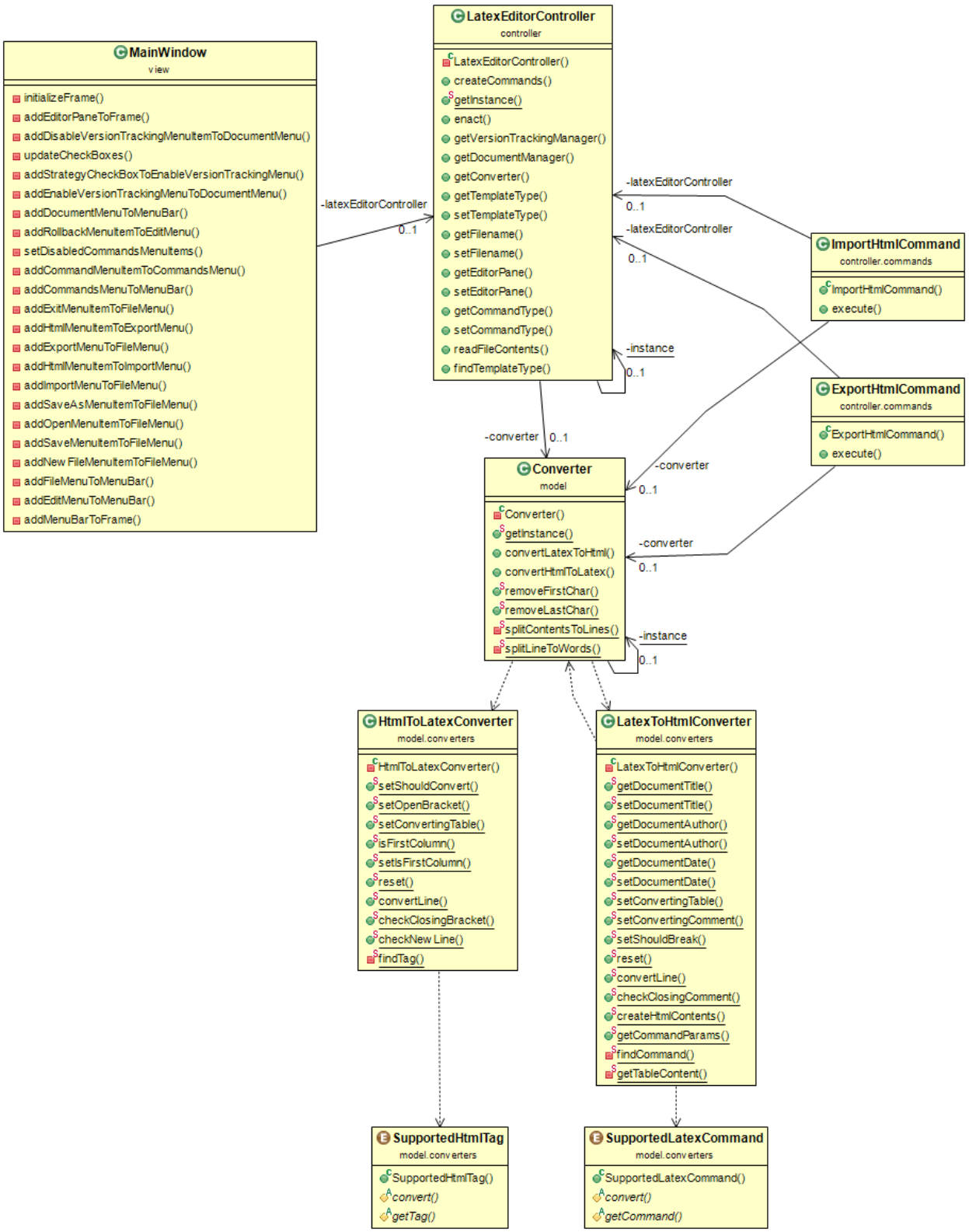
This class is similar to HtmlToLatexConverter. It is also placed in the model.converters sub-package. Its main purpose is to convert a line from Latex to HTML, while often taking into consideration factors that might come from another line. For instance, during table conversion. The main method of the class is convertLine() which, as its name suggests, is called from the Converter class and converts the strings of a line from Latex to HTML. The class has a helper method which iterates over the SupportedLatexCommand enum values to find whether a string corresponds to a Latex command. Moreover, the class has helper methods and boolean fields, regarding conversion of comments, tables and Latex command parameters.

ImportHtmlCommand class

This class is part of the controller.commands sub-package. It implements a command which reads the contents of a file on disk. Converts the contents to Latex using the Converter class. Checks if the Latex document corresponds to any of the template types supported by the application. And finally, sets the loaded Latex document as the current document of the application. In essence, the purpose of this command class is to import an HTML document. The command is called from the respective Import -> HTML MenuItem of the MainWindow GUI.

ExportHtmlCommand class

This class is similar to the ImportHtmlCommand class. It is also part of the controller.commands sub-package. It implements a command which fetched the current document of the application. Converts it to HTML using the Converter class. And finally, saves the converted HTML document at a specified location on disk. In essence, the purpose of this command class is to export the Latex document as an HTML document. The command is called from the respective Export -> HTML MenuItem of the MainWindow GUI.



UML class diagram with classes involved in HTML document import and export

CLASS-RESPONSIBILITES-COLLABORATIONS CARDS

In this section a brief description of each class is provided in terms of a CRC card. The cards can also be found in json format in the project contents in the respective folder.

LatexEditorController	
• Decouples view classes from model classes • Knows filename • Knows document template type • Stores commands • Enacts commands • Reads contents of loaded file • Finds template type of loaded file	• DocumentManager • VersionTrackingManager • Converter • CommandFactory

CommandFactory	
• Creates the command classes of the application	• -

CreateCommand	
• Creates a new document • Sets a new document as the current document	• LatexEditorController • DocumentManager

AddLatexCommand	
• Adds latex commands to current document • Adds latex commands to editor pane • Updates caret position of editor pane	• LatexEditorController • Strings

EnableVersionTrackingCommand	
• Enables version tracking	• VersionTrackingManager

DisableVersionTrackingCommand	
• Disables version tracking	• VersionTrackingManager

ChangeVersionTrackingStrategyCommand	
• Changes the currently enabled version tracking strategy	• VersionTrackingManager

EditCommand	
• Saves current document version based on the enabled version tracking strategy	• LatexEditorController • VersionTrackingManager • DocumentManager

RollbackCommand	
• Rolls back to previous document version • Updates editor pane contents	• VersionTrackingManager • LatexEditorController • DocumentManager

SaveCommand	
• Saves current document on disk	• DocumentManager • LatexEditorController

LoadCommand	
• Loads document from disk	• DocumentManager • LatexEditorController

ExportHtmlCommand	
• Signals latex to html document conversion • Saves the converted document on disk	• DocumentManager • LatexEditorController • Converter

ImportHtmlCommand	
• Loads document from disk • Signals html to latex document conversion • Creates new converted document	• DocumentManager • LatexEditorController • Converter

Converter	
• Converts latex document to html document • Converts html document to latex document	• LatexToHtmlConverter • HtmlToLatexConverter

HtmlToLatexConverter	
• Converts a line from html to latex	• -

LatexToHtmlConverter	
• Converts a line from latex to html	• Converter

Document	
• Knows contents of a document • Knows version ID of a document • Saves document to disk • Clones document • Changes version ID of a document	• -

DocumentManager	
• Knows the current document • Creates new document • Knows contents of document templates	• Document

VersionTrackingManager	
• Knows whether version tracking is enabled or not • Knows the current version tracking strategy • Enables version tracking • Disables version tracking • Changes version tracking strategy • Rolls back to previous document version	• StableStrategy • VolatileStrategy • StrategyFactory

StrategyFactory	
• Creates the version tracking strategy classes of the application	• -

StableStrategy	
• Stores the version history of a document on disk • Changes the version history of a document • Clears the version history of a document	• Document

VolatileStrategy	
• Stores the version history of a document in main memory • Changes the version history of a document • Clears the version history of a document	• Document

OpeningWindow	
• Initiliazes opening window GUI • * Serves as the application's entry point	• LatexEditorController

ChooseTemplate	
• Initializes choose template GUI	• LatexEditorController

MainWindow	
• Initializes main window GUI	• LatexEditorController

TEST CASES

Test cases have been developed for each of the application's user stories. To assist with the testing the EnvironmentSetup class has been developed in the respective test package the purpose of which is to simulate the application environment during runtime. Specifically, it initializes the various GUI components along with the application logic and simulates the actions that correspond to each user story. The test cases provide roughly a coverage of 90% of the source code.

EnvironmentSetup	
• Simulates the application environment during runtime	• OpeningWindow • ChooseTemplate • MainWindow • LatexEditorController • StableStrategy • VolatileStrategy

CRC card for the EnvironmentSetup class

The following test cases have been developed: (The test cases can also be found in json format in the respective project folder)

ID:	T1_V0
Test Class:	CreateCommandTest
Description:	Tests latex document creation
Pre-cond.:	Application successfully initialized and running on choose template GUI
Input:	Document template type
Output:	No output generated
Post-cond.:	Application transitioned to main window GUI with a newly created document based on the given template type
Class under test:	CreateCommand
Method(s) under test:	(1) execute

ID:	T2_V0
Test Class:	AddLatexCommandTest
Description:	Tests latex document creation
Pre-cond.:	Application successfully initialized and running on main window GUI
Input:	(1) Document template type (2) Latex command type
Output:	No output generated
Post-cond.:	Added the corresponding latex command to the current document
Class under test:	AddLatexCommand
Method(s) under test:	(1) execute

ID:	T3_V0
Test Class:	EnableVersionTrackingCommandTest
Description:	Tests version tracking enablement
Pre-cond.:	Application successfully initialized and running on main window GUI
Input:	(1) Document template type (2) Strategy type
Output:	No output generated
Post-cond.:	Version tracking enabled with the corresponding strategy type
Class under test:	EnableVersionTrackingCommand
Method(s) under test:	(1) execute

ID:	T4_V0
Test Class:	DisableVersionTrackingCommandTest
Description:	Tests version tracking disablement
Pre-cond.:	Application successfully initialized and running on main window GUI with enabled version tracking
Input:	(1) Document template type (2) Strategy type
Output:	No output generated
Post-cond.:	Version tracking disabled
Class under test:	DisableVersionTrackingCommand
Method(s) under test:	(1) execute

ID:	T5_V0
Test Class:	ChangeVersionTrackingStrategyCommandTest
Description:	Tests version tracking disablement
Pre-cond.:	Application successfully initialized and running on main window GUI with enabled version tracking and non-empty doc ument versions
Input:	(1) Number of document versions (2) Document template type (3) From-Strategy type (4) To-Strategy type
Output:	No output generated
Post-cond.:	Version tracking changed and document versions transfered to the newly enabled strategy
Class under test:	ChangeVersionTrackingStrategyCommand
Method(s) under te st:	(1) execute

ID:	T6_V0
Test Class:	EditCommandTest
Description:	Tests version tracking disablement
Pre-cond.:	Application successfully initialized and running on main window GUI
Input:	(1) Document template type (2) Input text
Output:	No output generated
Post-cond.:	Edited document version saved as current document
Class under test:	EditCommand
Method(s) under test:	(1) execute

ID:	T7_V0
Test Class:	RollbackCommandTest
Description:	Tests roll back to previous document version
Pre-cond.:	Application successfully initialized and running on main window GUI with enabled version tracking and non-empty document versions
Input:	(1) Number of document versions (2) Document template type (3) Strategy type
Output:	No output generated
Post-cond.:	Current document version rolled back through all previous document versions
Class under test:	RollbackCommand
Method(s) under test:	(1) execute

ID:	T8_V0
Test Class:	SaveCommandTest
Description:	Tests document save on disk
Pre-cond.:	Application successfully initialized and running on main window GUI
Input:	(1) Destination path
Output:	Tex file with the document contents in the test_files/tex directory
Post-cond.:	Application state intact
Class under test:	SaveCommand
Method(s) under test:	(1) execute

ID:	T9_V0
Test Class:	LoadCommandTest
Description:	Tests document load from disk
Pre-cond.:	Application successfully initialized and running on main window GUI or on opening window GUI
Input:	(1) Path of latex document on disk (2) GUI window
Output:	No output generated
Post-cond.:	Contents of the latex document on disk loaded as the current document of the application
Class under test:	LoadCommand
Method(s) under test:	(1) execute

ID:	T10_V0
Test Class:	ExportHtmlCommandTest
Description:	Tests html document export to disk
Pre-cond.:	Application successfully initialized and running on main window GUI
Input:	Destination path
Output:	HTML file with the converted latex document contents in the test_files/html directory
Post-cond.:	Application state intact
Class under test:	ExportHtmlCommand
Method(s) under test:	(1) execute (2) convertLatexToHtml

ID:	T11_V0
Test Class:	ImportHtmlCommandTest
Description:	Tests html document import from disk
Pre-cond.:	Application successfully initialized and running on main window GUI
Input:	(1) Path of html document on disk
Output:	No output generated
Post-cond.:	Contents of the html document on disk converted to latex and loaded as the current document of the application
Class under test:	ImportHtmlCommand
Method(s) under test:	(1) execute (2) convertHtmlToLatex